Global Localization of Autonomous Underwater Vehicles Using Visual Odometry

Honors Undergraduate Research Thesis

Presented in Partial Fulfillment of the Requirements for Graduation with Honors Research Distinction in the Department of Mechanical and Aerospace Engineering of The Ohio State University

By

Hao Yu

Department of Mechanical and Aerospace Engineering

The Ohio State University

2021

Thesis Committee:

Professor Ayonga Hereid, Ph.D., Advisor

Professor Saeedeh Ziaeefard, Ph.D.

Abstract

Path planning and navigation are two of the main challenges for achieving autonomous driving. At the same time, localization has been considered as the footstone for path planning and navigation because the odometry information of a vehicle is required for the high-level planning algorithms. For autonomous underwater vehicles (AUV), expansive navigation sensors, such as deepsea sonar, have usually been utilized to get the location of the robot underwater. This project, however, is intended to introduce the potential application of real-time underwater visual odometry (VO) algorithms based on data retrieved from a monocular camera on both simulated environments and real underwater datasets. Due to the outbreak of the pandemic, this project is primarily developed and tested based on the platform of an underwater simulator named UWSim. Since the working environment for underwater robots is not always perfect, different feature detection algorithms were performed on real open-source underwater footage to determine the most suitable image processing algorithm for this project. The proposed visual odometry algorithm is based on state-of-the-art feature detection algorithms, feature tracking techniques based on optical flow, and projected geometry theories. Comparisons of the ground truth odometry of a simulated underwater robot and the odometry calculated from the visual odometry algorithm will be presented.

This thesis is dedicated to my father Yu Zhenglin and my mother Che Xin.

My most enormous pride is becoming your pride.

Acknowledgments

I was truly fortunate that many notable people had offered me help, encouragement, and advice during the journey of working on this thesis.

Firstly, I would like to thank my research advisor, Professor Ayonga Hereid, who offered me the opportunity to work in his lab, the Cyberbotics Lab, and support me in all directions in this project. I sincerely appreciate his guidance in the fundamental knowledge of robotics and his patience for this project. This project would never happen without his belief in me.

Secondly, I would like to express my appreciation for the support from my lab colleagues in the Cyberbotics Lab. I would like to thank Guillermo Castillo-Martinez for suggesting approaches to conduct this research. I would like to thank Jack Beokhaimook for his help with the RANSAC functions and transformations. I would also like to thank Archit Rede for his inspiration in computer vision.

In addition to my lab colleagues, I would like to thank Ted Sender, Ph.D. candidate at the University of Michigan, for guiding me into the field of robotics. I would like to thank the Underwater Robotics team at the Ohio State University for providing me the chance to explore my passion for underwater robotics. Thank you to Professor Saeedeh Ziaeefard, who encouraged me to work on this project. Lastly, I would like to thank the Department of Mechanical and Aerospace Engineering for awarding me the Undergraduate Research Scholarship.

Table of Contents

List of Tables

Table **Page**

List of Figures

Abbreviations

AUV	Autonomous Underwater Vehicle
DoF	Degrees of Freedom
DVL	Doppler Velocity Log
FAST	Features From Accelerated Segment Test
GFTT	Good Features To Track
IMU	Inertial Measurement Unit
LKT	Lucas-Kanade Tracker
ORB	Oriented FAST and Rotated BRIEF
PID	Proportional Integral Derivative
RANSAC	Random Sample Consensus
ROS	Robot Operating System
SIFT	Scale-invariant Feature Transform
SURF	Speeded Up Robust Features
TF	Transform
VO	Visual Odometry
VSLAM	Visual Simultaneous Localization and Mapping

Chapter 1: Introduction

1.1 Background and Motivation

As we know, about 71 percent of the Earth's surface is covered by water. At the same time, oceans hold nearly 97 percent of water on Earth. Despite the size and the impact on all living creatures of the ocean, it remains a mystery because more than 80 percent of the ocean has never been explored by humans [22]. The tremendous pressure and the bearing witness underwater have made the tasking of exploring underwater environments extremely difficult [11]. However, the invention of autonomous underwater vehicles (AUV) has made breakthroughs in discovering oceans and made many underwater investigations possible.

An AUV is a pre-programmed unmanned submersible machine that operates without any real-time instructions from human operators. The machine classified as the first AUV is SPURV, which was invented in 1957 by the Applied Physics Laboratory at the University of Washington. The chassis of SPURV was made of aluminum with a torpedo-like shape driven by a screw. Since the lack of computational power, this vehicle was controlled by acoustic communications, while it was still capable of staying 10,000 feet underwater for four hours. After successfully operating as a reliable tool to collect data for oceanographic research for 22 years, the University of

Washington replaced SPURV with a more advanced AUV based on the platform of SPURV in 1979, the SPURV II. Unlike its predecessor, SPURV II was equipped with many sensors and controlled by a computer mounted inside. With the development of technology, possibilities of AUVs have been highly expanded in the next few decades. In 1998, SAUV was created by the Russian Institute of Marine Technology Problems in cooperation with the Autonomous Undersea System Institute. This vehicle's significant improvements were its solar panels and its capability to navigate based on the global position system (GPS) [25].

In recent years, more modern AUVs equipped with a long list of sensors have been manufactured and deployed for marine research, including oceanographic surveys, demining, and marine biology research. Among the fields of research for AUVs, localizating the vehicle underwater is one of the most challenging topics because of the physical properties of water. Some approaches for estimating the position of a vehicle or a robot above water, such as radio communications, will not function as desired in subsea scenarios because of the strong electromagnetic interferences of the medium [15]. Most of the modern AUVs are relying on sensors ranging from magnetic compass, pressure depth sensor, Doppler velocity log (DVL), inertial measurement unit (IMU), to acoustic modems, cameras, and sonars to solve this problem [16]. In most cases, utilizing sonars to localize the vehicle underwater is considered as the best solution. However, the price of an undersea sonar is relatively high compared to other sensors listed. Besides, there are many drawbacks associated with implementing a sonar, including but not limited to its low data rate and its high latency due to the slow speed of sound in water. Inertial-based location estimation based on data from

sensors, for example, DVL or IMU, is an alternative method to the acoustic-based localization, while it suffers from the incrimination of linearization errors [17].

This thesis's motivation is to figure out a method that enables the user to get the global localization of an AUV within a tight budget. As a result, this thesis establishes a robust visual odometry (VO) algorithm based on a monocular camera. The term visual odometry was first introduced by Nister et al. in 2004 [14]. They defined visual odometry as a process of estimating the motion of a single moving camera based on video input. The proposed VO algorithm in this thesis has a similar structure to the one defined by Nister et al., while at the same time, the VO algorithm in this thesis changes and optimizes the original VO algorithm. The proposed algorithm will not require the developers to spend more than $1,500 to purchase a sensor for localizing a robot underwater, and it is portable for various structures of AUVs. This algorithm was developed and tested on a simulator, while it is still considered robust under turbid, real-world underwater environments.

1.2 Outline of the Thesis

The organization of the following chapters is shown below.

Chapter 2 will focus on the problem formulation. A high-level view of the VO algorithm introduced by Nister et al. [14] will be covered. Relative works conducted by other researchers on VO algorithm will also be introduced.

Chapter 3 will focus on the methodology of this research. Firstly, the platform for this research, UWSim, will be introduced. Work on this platform will be revealed, and inputs from this platform to the VO algorithm will be shown. Secondly, the

framework of the proposed VO algorithm will be addressed. The proposed algorithm will be broken down into steps, and each step will be described in detail.

Chapter 4 will focus on presenting results from testing the VO algorithm in the simulator and discussing the accuracy rate of the algorithm in those tests. Four rounds of tests were conducted to evaluate the robustness of the proposed VO algorithm under different robot moving cases.

Chapter 5 will include the conclusions from this thesis, a summary of the advantages and limitations of the proposed algorithm, and suggestions for future work.

Chapter 2: Problem Formulation

In this chapter, we introduce the platform that was used for this research. The model of the simulated robot and the specifications of the monocular camera are denoted in detail. Also, a high-level view of the Nister et al.'s VO algorithm [14] and literature reviews of other VO algorithms and their accuracy rates are included.

2.1 Introduction of the Platform

We discussed in the previous chapter that the major platform for developing and testing the proposed algorithm is UWSim, which is an underwater simulator for marine robotics research. The simulator provides visualizations of an underwater scenario that can be configured by standard modeling software. In the standard simulator, there is a single simulated underwater robot, a model of the Girona 500 AUV [5], equipped with a range camera, a monocular camera, an IMU, and a DVL. The local geodetic coordinate system defined in the standard simulator is East-North-Up (ENU), which means the X direction in this simulator corresponds to East, the Y direction corresponds to North, and the Z direction corresponds to Up, as shown in Figure 2.1.

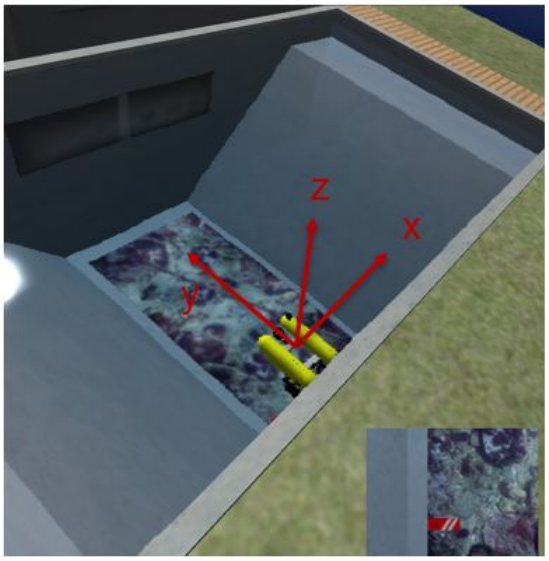

Figure 2.1: Screenshot of the UWSim simulator

There are many reasons for choosing UWSim. Four primary reasons are:

- This simulator works with the robot operating system (ROS) and Ubuntu 18.04, which simplifies the process of getting the ground truth odometry of the simulated robot and allows the proposed algorithm to work in all ROS platforms.

- There is an inbuilt data navigator that is accessible to users [18], and it helps expedite the development of controllers for the simulated robot.

- The monocular camera mounted to the simulated robot is facing downwards, which is the ideal position for the proposed VO algorithm.

- Calibration of the monocular camera is provided on the UWSim website [1] and it is accessible to users by subscribing the *uwsim/camera1_info* ROS topic. Some

major parameters from the calibration of the camera are shown in Table 2.1, which fulfills the intrinsic matrix of the camera K, where

$$\mathrm{K} = \begin{bmatrix} fx & 0 & x0 \\ 0 & fy & y0 \\ 0 & 0 & 1 \end{bmatrix}, \mathrm{K} \in \mathrm{SO}(3)$$

Parameter	Value
focal length, fx	257.986
focal length, fy	257.341
x position of principle point, $x0$	120
y position of principle point, $y0$	160

Table 2.1: Caption

2.2 Literature Review

When the original visual odometry algorithm was introduced in 2004, it was composed by three parts, which are feature detection, feature matching, and robust estimation. In the feature detection part, it applied the Harris Corner Detector [9] on the incoming images to detect point features. Next, in the feature matching part, it used mutual consistency check to match features detected from two consecutive frames. Eventually, the Five-point algorithm was utilized to determine the essential matrix and decompose it to get the transformation matrix. Many other researchers have optimized the initial VO algorithm following their own approaches. Among researchers who rely on solely one monocular camera, Van Hamme et al. altered the feature detection method and reported to get an error rate larger than 8.5% in an 800 meters test [24]; Zhang et al. changed the mutual consistency check tracking approach to Lucas-Kanade-Tomasi (KLT) feature tracking, and the error rate for

their algorithm was smaller than 1% in a test of 1000 meters [27]. According to [2], a review of popular VO algorithms by Aqel et al. in 2016, an error rate of 3% is common in most research in VO algorithm. Therefore, the goal of this research is achieving an accuracy rate higher than 97%.

Chapter 3: Methodology

In this chapter, we cover the operations we performed on the platform, UWSim, as we introduced in the last chapter, to get the inputs for the VO algorithm. Next, we broke the proposed VO algorithm into four steps, and each step is explained in detail. The changes and optimizations we added to the original VO algorithm [14]. are highlighted.

3.1 System Configuration

The work on this platform started by looking into the structure of the simulated robot. After thoughtful observations on the transformation (TF) tree in ROS (part of the TF tree is shown in Figure 3.1 and the XML configuration Figure 3.2 published on the UWSim official website [1], we noted that the name of the link of the monocular camera is *bowtech1*. We also found the static transformation between the link of the monocular camera and the *girona500/part_0* link. The transformation can be represented by

$$R = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}, R \in SO(3) \tag{3.1}$$

$$t = \begin{bmatrix} -0.2 \\ -0.1 \\ 0 \end{bmatrix}, t \in \mathbb{R}^3 \tag{3.2}$$

where R is the rotation matrix, which denotes the orientation of the camera link relative to the *girona500/part_0* link; t is the translation vector, which shows the linear distance from the part_0 link in the simulated Girona 500 AUV to the position of the camera in 3D. In this project, we choose the *base link* of the robot as the local coordinate of the robot. We get the ground truth odometry of the simulated robot in the world coordinate by subscribing to the transformation between the *base link* and the *world* frame.

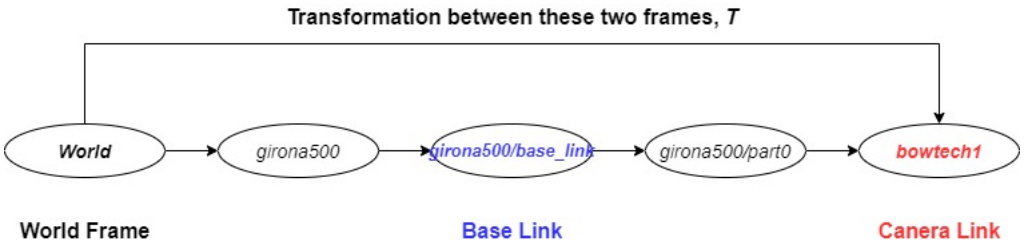

Figure 3.1: Part of the TF tree in UWSim

After successfully getting the odometry of the simulated robot and the position of the camera, we then moved on to establish a kinematic velocity controller for controlling the robot. The development of the velocity controller has been greatly simplified thanks to the built-in ROSOdomToPAT interface available in this simulator. The next task for working with the platform was creating a position controller. This position controller is a proportionalinertialderivative (PID) controller based on the velocity controller we stated above and position commands.

```
<virtualCamera>
  <name>bowtech1</name>
  <relativeTo>part0</relativeTo>
  <resw> 320 </resw>
  <resh> 240 </resh>
  <position>
    <x>-0.2</x>
    <y> -0.1 </y>
    <z> 0 </z>
  </position>
  <orientation>
    <r>0</r>
    <p>0</p>
    <y>1.57 </y>
  </orientation>
</virtualCamera>
```

Figure 3.2: XML configuration of the monocular camera

Due to the nature of monocular visual odometry, we cannot decide an exact scale for the translation from the continuous image frames. If no scale is provided to the VO system, estimated positions of the camera will be ratio values without unit. As a result, we need to obtain the scale value, which is essentially the moving speed of the camera from some extra resources. While implementing VO algorithms in real-life, the scale information is usually calculated based on data retrieved from some sensors, for example, wheel counter or IMU. In our case, however, we can directly get the scale value from the velocity controller since we are only using the kinematic mode of the simulator. Namely, the simulated robot moves at the exact velocity we command. At this point, we have all inputs that the VO algorithm needs from the simulator, which are:

1. Images captured by the monocular camera mounted on the simulated robot, I. Image inputs to this VO algorithm were acquired from the *uwsim/camera1* ROS topic, which publishes a *sensor_msgs/Image* type message.

2. The absolute speed of the robot moving on the xyplane, s. This speed was obtained directly from the velocity controller.

3. Calibration information of the camera. In this simulator, the monocular camera's calibration information was stored in the *uwsim/camera1_info* ROS topic, which publishes a *sensor_msgs/Camerainfo* type message. We will get the intrinsic matrix of the camera, K, the height and the width of images captured from that message.

4. Transformation between the *girona500/base_link* and the *world* frame. This transformation will give us the ground truth odometry of the simulated robot in the world coordinate.

5. Transformation between the camera link and the world frame, T, which we will use in the camera position estimation part to convert the camera position from the local coordinate to the world coordinate.

6. Starting position of the robot in the local coordinate, p_0. It is default to the origin of the local coordinate based on the camera link when the simulator starts.

3.2 Visual Odometry Algorithm

In this part, we cover details of the proposed VO algorithm. First, the framework of the algorithm is presented, the pseudo-code of the algorithm is also shown. Next, four inner related parts, which are feature detection, feature tracking, essential matrix calculation, and camera position estimation, that compose the VO algorithm will

be explained. We focus on changes and optimizations we added to the initial VO algorithm.

3.2.1 Framework of the algorithm

In opposition of the approach presented in [14], this proposed VO algorithm applied the Kanade–Lucas–Tomasi (KLT) method to perform feature tracking [23]. In addition, we optimized the feature detection and tracking portion to make sure the system is more robust in turbid, dynamic underwater environment, and ensure that all the features tracked are in the view of the camera. We also modified the approach to estimate the position of the camera by adding a filter to the system to reduce the drifts from the visual Results. Algorithm 1 shows the pseudo-code of the proposed VO algorithm.

Algorithm 1: Pseudo-code of the proposed algorithm

input : Images captured I^t, I^{t+1}; Scale s; intrinsic matrix K;
transformation camera link to world T; starting position p_0
output: Odometry of the robot in the world coordinate

initialization;
$M^t \leftarrow$ convertRGBToGrayscale(I^t);
$pt_1 \leftarrow$ featureDetection(M^t);
while *system isn't shutdown* **do**
  $M^{t+1} \leftarrow$ convertRGBToGrayscale(I^{t+1});
  $pt_2 \leftarrow$ featureTracking(M^t, M^{t+1}, pt_1);
  $fpt_1, fpt_2 \leftarrow$ filterPoints(pt_1, pt_2);
  $E \leftarrow$ findEssentialMat(fpt_1, fpt_2, K);
  $R, t \leftarrow$ getTransformation(fpt_1, fpt_2, E);
  $cam\_pose \leftarrow$ getCamPose(R, t, s, p_0);
  $robot\_pose \leftarrow$ getRobotPose(cam_pose);
  $robot\_pose \leftarrow cam\_pose$;
  $M^t \leftarrow M^{t+1}$;
  $pt_2 \leftarrow fpt_1$;
end

3.2.2 Feature Detection

Feature detection is the first step for the VO algorithm. After the system receives the image inputs and convert it to grayscale as line 2 noted in Algorithm 1, the system will start a feature detector and perform feature detection on the incoming grayscale images. Since the research is conducted in a simulator, water inside it is observed to be clear and crystal. However, the real-world underwater environment is identified to be more turbid and dynamic. Inspired by [7], we decided to test some feature detection algorithms on a real-world underwater dataset to determine which algorithm works the best in turbid underwater scenarios. There are numerous opensource feature detection algorithms available, we chose five popular algorithms for testing, which are SIFT [12], SURF [4], FAST [10], GFTT [21], and ORB [20].

This real underwater dataset [6] was utilized to test the above five algorithms. While testing the algorithms, no constrains for the detectors, for example, maximum numbers of features to be extracted, or the minimum distance for two features, were set.

3.2.3 Feature Tracking

Feature tracking is a critical portion for this research because we need to associate points extracted from consecutive images in order to generate the trajectory of the robot underwater. In this project, we chose the KLT feature tracker [23] to match keypoints extracted from the previous frame using feature detection algorithms with keypoints from the incoming frame. The idea behind the tracking part of KLT is it analyzes the optical flow with the Lucas-Kanade implementation. The process for feature tracking in this system is noted in line 6 in Algorithm 1.

While since KLT relies on optical flow for tracking, it is not robust in dynamic underwater environment: features for tracking might get lost if some moving objects block the view of the camera. To find a solution that will alleviate the drawbacks of KLT, we added a filter in the feature tracking part as noted in line 7 in Algorithm 1. The filter operates in two steps. The first step checks the positions of keypoints extracted and it will delete points with negative position values. The purpose of the first step is to make sure that all the input points for tracking are in the view of the camera. The second step checks for the number of keypoints extracted for tracking, and if the number is smaller than a threshold, which is a constant set by the RANSAC function for calculating the essential matrix that we will talk about in the next part, it will restart the feature detection on the images. The purpose for the second step is to make sure that there are enough samples for the random sampling function [8].

3.2.4 Essential Matrix Estimation

Essential matrix is required to estimate the relative camera motion from a stream of image frames. In this research, the start-of-the-art *Five-point algorithm* proposed in [13] was utilized to calculate the essential matrix as noted in line 8 - 9 in Algorithm 1. The input to the *Five-point-algorithm* are five keypoints we tracked in the previous part, while instead of randomly choosing five matching keypoints from the set of features, we apply the RANSAC function to get rid of the outliers. Based on [19], we know the number of samples must satisfy the inequality (3.3), where N is number of samples, p is probability of success, e is proportion of outliers, and s is good sample output. In our case, we know s is 5; in addition, we assume p to be 0.99 and e to be 50%. Plugging in the above values into inequality (3.3), we get $N > 146$, which is

set to the threshold of minimum features tracked between two consecutive images in the previous part.

$$N \quad > \quad \frac{log\,(1 \ - \ p)}{log\,[1 - (1 - e)^s]} \tag{3.3}$$

After performing the RANSAC function we described above, we will two sets of matching keypoints, which are $[x_n^t \quad y_n^t]$ and $[x_n^{t+1} \quad y_n^{t+1}]$, where $n = 1, 2, 3, 4, 5$. Next, we combine the two sets of points into two matrices, which are

$$pos_n^t = \begin{bmatrix} x_n^t & y_n^t & 1 \end{bmatrix}, n = [1, 2, 3, 4, 5] \tag{3.4}$$

$$pos_n^{t+1} \quad = \quad \begin{bmatrix} x_n^{t+1} & y_n^{t+1} & 1 \end{bmatrix}^T, n = [1, 2, 3, 4, 5] \tag{3.5}$$

In [13], it pointed that

$$pos_n^t \ \times E \ \times \ pos_n^{t+1} \ = \ 0 \tag{3.6}$$

where E is the essential matrix, and Et needs to satisfy the constrain of

$$\det(E) \ = \ 0 \tag{3.7}$$

which leads to the cubic constrain on the essential matrix shows in Equation (3.8).

$$EE^TE \ - \ \frac{1}{2}\text{trace}\left(EE^T\right)E \ = \ 0 \tag{3.8}$$

After solving the equations for the essential matrix listed above, the system will get the estimation of the essential matrix, E. Next, we need to decompose the essential

matrix to get the rotation matrix and the translation vector. In this project, we used the singular value decomposition method to decompose the essential matrix. Below shows the process of decomposing the essential matrix using the built-in MATLAB *svd*() function [3].

$$[U,\ S,\ V] \quad = \quad \text{svd}(E) \tag{3.9}$$

$$W \quad = \quad \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$R \quad = \quad U \cdot W \cdot V^T \tag{3.11}$$

$$t \quad = \quad U(:,\ 3),\ t \in \mathbb{R}^3 \tag{3.12}$$

where R in Equation (3.11) is the rotation matrix, W in Equation (3.10) is the rotation matrix of the camera corresponding to the *world* frame, and t in Equation (3.12) is the translation vector.

3.2.5 Robot Position Estimation

After estimating the essential matrix and getting the rotation matrix, R, and the translation vector, t, we calculate the motion of the robot by first getting the position of the camera and then figuring out the robot's position based on the position of the camera and the transformation between the camera link and the base link of the robot. Line 10 in Algorithm 1 shows the first step, and line 11 shows the second step.

First, we find out the transformation matrix, T, based on the rotation matrix and translation vector,

$$T \quad = \quad \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \; T \; \in \; SE(3) \tag{3.13}$$

Then we apply the transformation on the starting position of the camera, p_0, and get the updated position of the camera

$$cam\_pose \quad = \quad T \; \cdot \; p_0 \; \cdot \; s \tag{3.14}$$

where s is the scale of the camera motion. As we denoted in Section 3.1, in this project, the scale is the speed of the robot at that particular moment. To reduce the errors from drifts that rooted in feature tracking and estimation of the essential matrix, we added a filter in this step aiming to get rid of the noise. This filter is similar to the naive algorithm of the median filter [26], and the pseudo-code of the filter is presented in Algorithm 2.

Algorithm 2: Pseudo-code of the filter on calculated positions of camera

input : Window of 5 consecutive estimation of camera's position *data*
output: Filtered camera's position *cam_pose*

```
initialization;
sort(data);
data.pop(0);
data.pop(lengthofdata);
cam_pose ← avg(data);
```

After getting the filtered position of the camera, we need to transform that position to get the global localization of the robot. Assume the transformation between the camera frame and the world frame is T_world, we apply Equation (3.15) to get the desired localization.

$$robot\_pose \quad = \quad T\_world \; \cdot \; cam\_pose \tag{3.15}$$

where *cam_pose* is the result from the first step. We publish the output, *robot_pose*, by a ROS pubsliher. While the VO algorithm has not reached the end yet. We need

to assign the copy of the second grayscale image to the parameter of the first grayscale image, and assign the copy of the tracked points in the second image to the parameter of the tracked points of the first image to ensure that the system is able to function with a stream of images as inputs.

Chapter 4: Results and Discussions

In this chapter, we present the tests we conducted to test the feature detection algorithms and the whole VO system. Plots that visualize the results we collected from these tests are included in this chapter. We also summarize and discuss the results from each test.

4.1 Evaluation of Feature Detection Algorithms

As we introduced in the previous chapter, five popular feature detection algorithms, which are GFTT, SIFT, SURF, FAST, and ORB are subject to be tested on an underwater image dataset with different turbidities levels. As shown in Figure 4.1, results from applying the above five feature detection algorithms without any limitations on the maximum keypoints extracted on this particular image are presented (Figure 4.2- 4.6).

A table which lists all the numerical values of the keypoints extracted on the entire dataset is also shown in Table 4.1. To better visualize the results of the feature detection test, a plot is also presented Figure 4.7.

Figure 4.1: Real underwater image, turbidity level 12

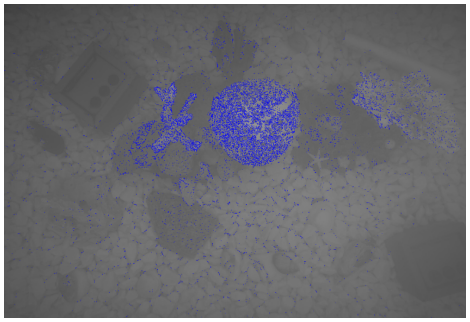

Figure 4.2: Result of performing GFTT

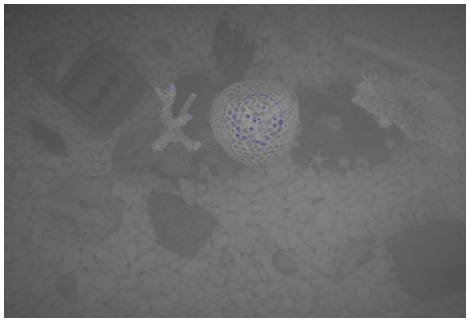

Figure 4.3: Result of performing SIFT

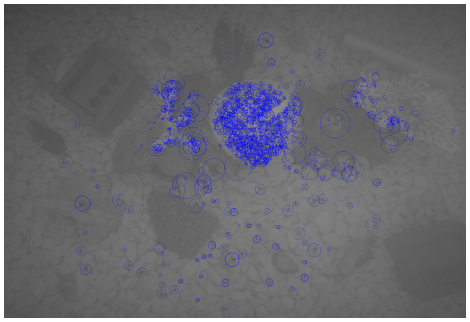

Figure 4.4: Result of performing SURF

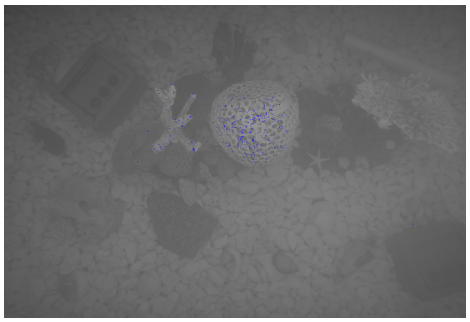

Figure 4.5: Result of performing SIFT

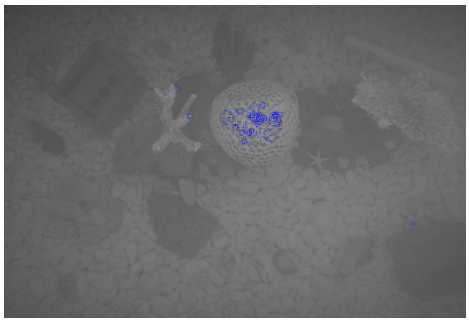

Figure 4.6: Result of performing ORB

Test of Feature Detection Algorithms					
Turbidity Level	GFTT	SIFT	SURF	FAST	ORB
1	50896	6572	24339	12867	18114
2	43560	8865	26153	18385	25556
3	31726	6650	22522	14016	18989
4	30326	5163	19620	11098	14041
5	26388	3784	16281	8766	10152
6	22310	2281	12286	5807	5663
7	22914	1932	10081	5773	4951
8	22740	767	6853	3099	1965
9	21125	616	5753	2636	1574
10	16739	287	3709	1471	699
11	21195	94	2205	656	215
12	23121	37	1545	347	78
13	20565	5	670	47	2
14	15246	3	526	34	3
15	13636	0	177	2	1
16	5965	0	98	1	1
17	4926	0	35	1	1
18	3885	0	16	0	0
19	749	0	3	1	2
20	5649	0	0	1	0

Table 4.1: Number of keypoints detected using different algorithms on various turbidity levels

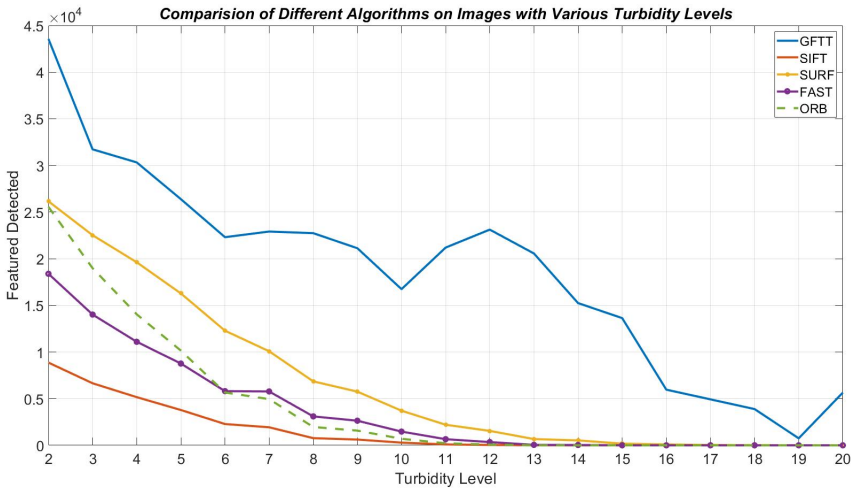

Figure 4.7: Visualization of results from the feature detection test

From the table and the plot above, without any limitations on the feature detectors, GFTT has the potentiality of extracting the greatest number of features from a turbid underwater image. SURF is the second best regarding the points extracted on this dataset among the five algorithms. ORB is assumed to function better than FAST in less turbid environment, however, when detecting features on images with more noise, FAST works better than ORB. SIFT is the worst among the listed feature detection algorithms in this test. Therefore, GFTT is considered to the most exceeding feature detection algorithm among all algorithms tested, and it will be applied in the feature detection step in the final VO algorithm.

4.2 Results from Linear Motion Tests

In Chapter 2, we introduced the platform for this research and explained the VO algorithm proposed. In order to get clear of the performance of the proposed algorithm, we conducted four rounds of tests in the simulator, collected data from each test, and created the Equation (4.1) to find out the accuracy rate of the algorithm under different robot's moving cases. Note that since North-East-Down (NED) world coordinate system are more occasionally used in the fields of AUV, we converted the East-North-Up (ENU) coordinate system used in the simulator to NED while recording data during tests and revealing the results below.

$$accuracy\ rate = \frac{total\ distance\ -\ drift\ distance}{total\ distance}\ \times\ 100\% \tag{4.1}$$

For the first linear motion test, we decided to perform an one degree of freedom (DoF) test. In this test, we commended the robot to move in x direction for 4.5 meters, and we recorded the data of the ground truth odometry of the robot and the trajectory of the robot generated by our VO algorithm. The ground truth odometry and the visual result are shown in Figure 4.8 and Figure 4.9.

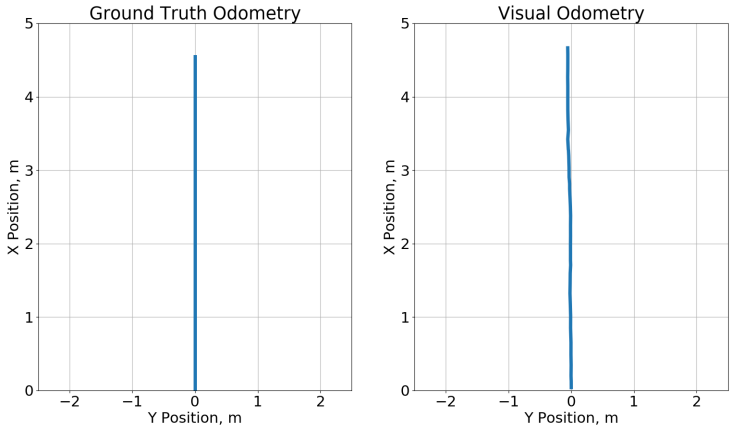

Figure 4.8: Visualization of results from the 1 DoF linear motion test, separated

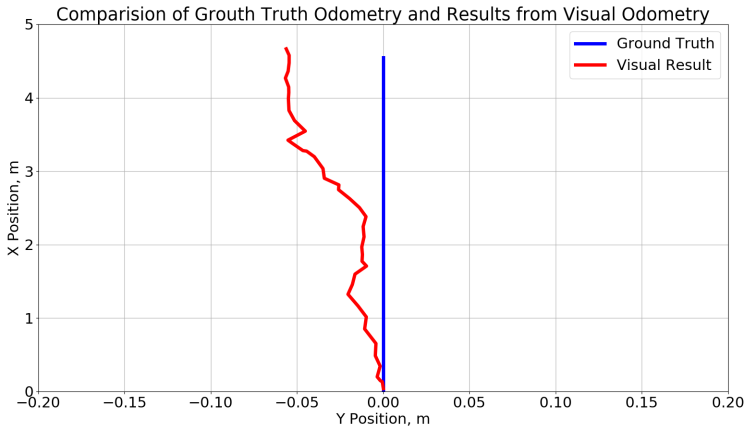

Figure 4.9: Visualization of results from the 1 DoF linear motion test, combined

By analyzing the data collected in this test, the total drift in this test was calculated to be 0.128 meters, the total distance traveled by the robot was 4.50 meters.

Plugging in these numbers into Equation (4.1), we found the accuracy rate for the 1 DoF test was 97.2%.

In the second test, we commended the robot to move in the positive x direction for 3.5 meters, then move in the positive y direction for 2.0 meters, move in the negative x direction for 3.5 meters, and move in the negative y direction for 2.0 meters. The robot moved in two directions, x and y direction, in this test, which made this test a 2 DoF test. Similar to the 1 DoF test, Figure 4.10 and Figure 4.11 visualize the results generated in this test. Based on the data collected in the 2 DoF test, the total drift was calculated to be 0.281 meters, the total distance travled by the robot in this test was 11.2 meters, which led to an accuracy rate of 97.4% for the VO algorithm using Equation (4.1).

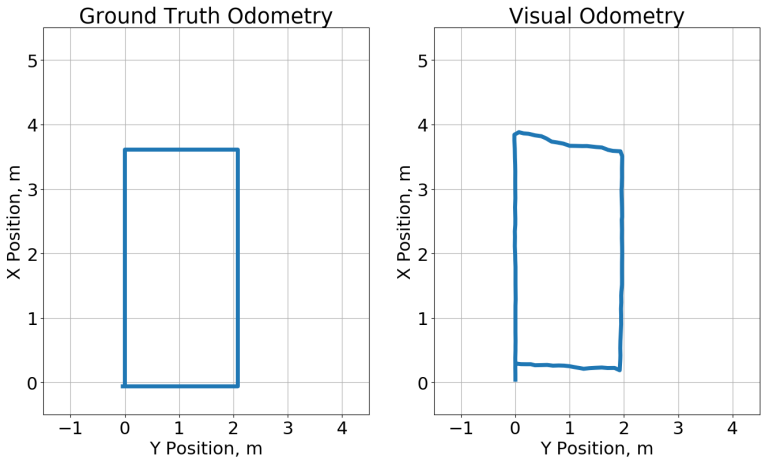

Figure 4.10: Visualization of results from the 2 DoF linear motion test, separated

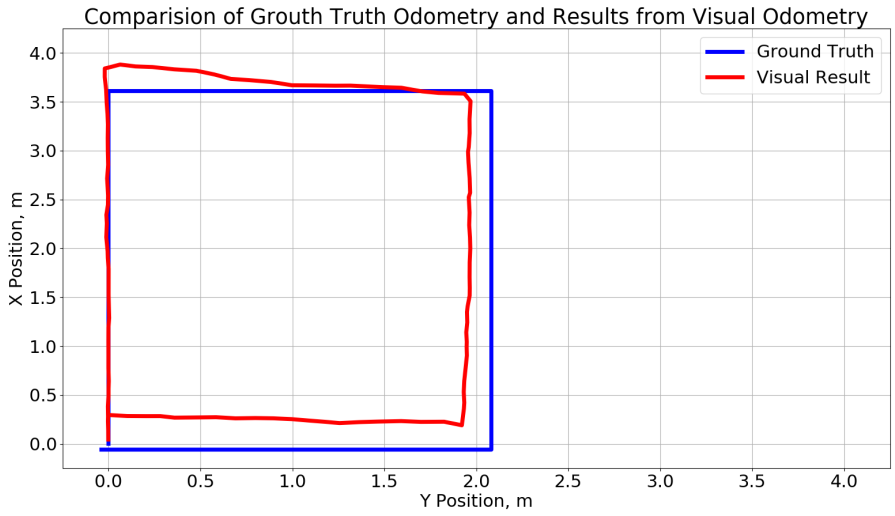

Figure 4.11: Visualization of results from the 2 DoF linear motion test, combined

4.3 Results from Rotation Motion Test

After conducting the linear motion tests, results from applying the VO algorithm while the robot is rotating were also collected. In the rotation motion test, the simulated robot was moving in random x, y directions and rotating randomly around the z axis. Visualization of the ground truth trajectory and the calculated trajectory of the robot based on the VO algorithm are presented in Figure 4.12 and Figure 4.13. The total drifted distance in this test was 1.19 meters, and the total distance travled by the simulated robot was 7.35 meters. Plugging the above values to Equation (4.1), we found the accuracy rate for this test was 83.8%.

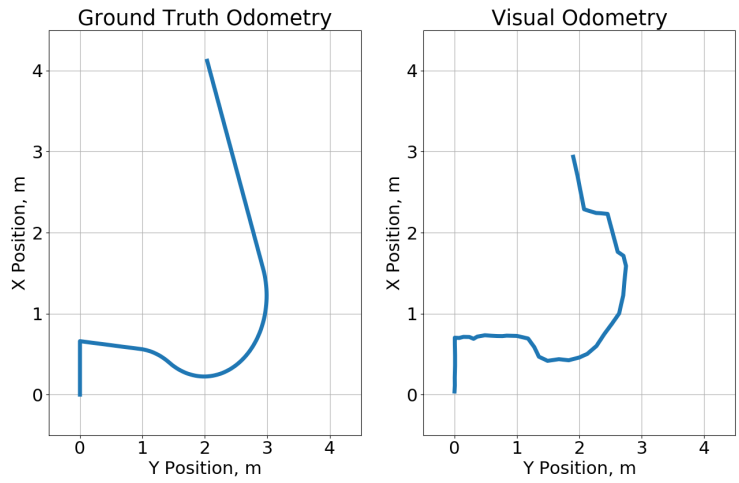

Figure 4.12: Visualization of results from the 3 DoF rotation motion test, separated

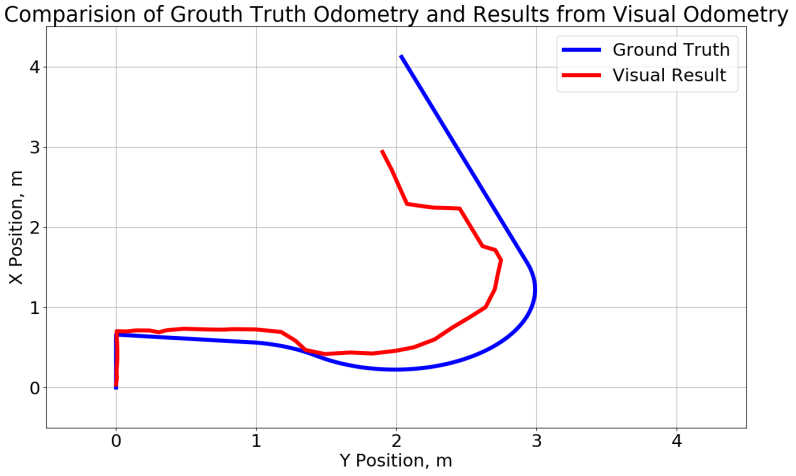

Figure 4.13: Visualization of results from the 3 DoF rotation motion test, combined

The final test was an autonomous driving test. We first drove the robot to $[4.0 \quad 1.5 \quad -2.0]^T$ manually based on the position controller. Then we commended the robot to move to $[1.5 \quad 0.25 \quad -2.0]^T$ automatically using the visual odometry as the localization approach for the position controller. The ground truth trajectory of the robot and the result from the visual odoemtry in this test is presented in Figure 4.14 and Figure 4.15. The total drift in this test was found to be 0.91 meters, and the total distance traveled by the robot was 8.35 meters. Plugging in these values into Equation (4.1), we found the accuracy rate in this test is 98.9%.

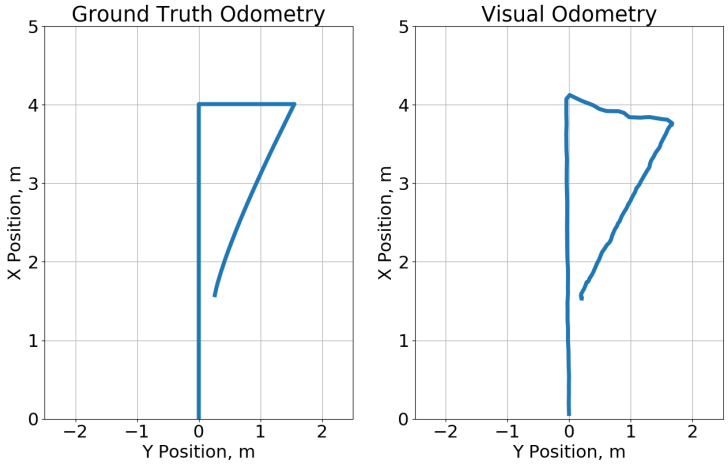

Figure 4.14: Visualization of results from the autonomous driving test, separated

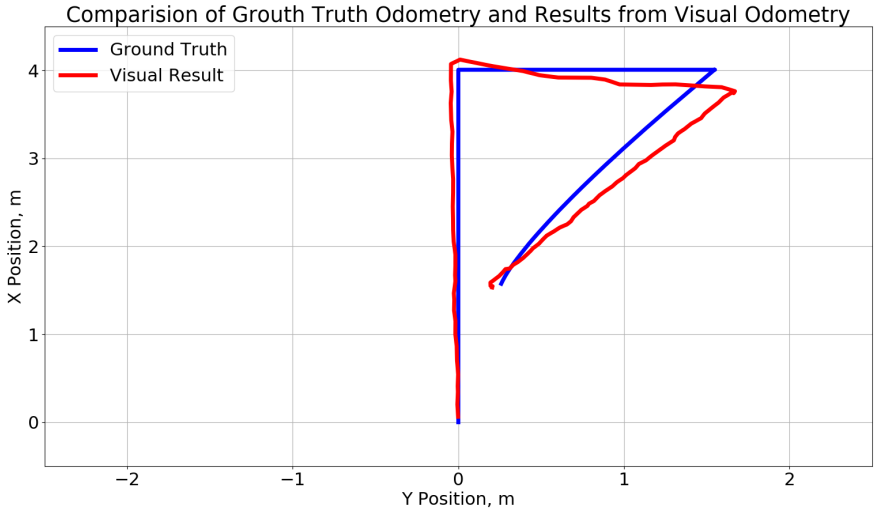

Figure 4.15: Visualization of results from the autonomous driving test, combined

4.4 Evaluation of Results from Robot Motion Tests

After conducting the four rounds of tests and calculating the accuracy rate for each test, we built this table Table 4.2. By observing the table, we found that the accuracy rate for the robot moving in straight lines are relatively high. The average accuracy rate for tests with less than 2 DoF robot's motion is 97.3%, which has a 2.7% error rate and thus it is assumed to be acceptable for estimating the motion of the robot compared to the 3% threshold we set in chapter 2. While consider the fact that due to the constrain of the simulator, the total distance traveled for the robot in those two cases are tremendously short compared to the tests of VO algorithm mentioned in chapter 2. Also, due to the nature of the VO algorithm, it will accumulate errors from estimation drifts with time. As a result, we assumed that theoretically, the accuracy

rate of performing this VO algorithm in real AUV working scenarios is smaller than the values listed in Table 4.2.

For the 3 DoF rotation motion test, however, the final accuracy rate, 83.8%, doesn't pass the minimum accuracy rate we set in chapter 2. Potential reasons we concluded for the low accuracy rate in this test are listed below:

- The filter on the raw camera position data creates a high latency of getting the final camera position.
- The feature tracker cannot handle rotation motions. It would consider rotation motions as linear translation.
- This system lacks a stable method for estimating the position of the camera.

In summary, results from these tests indicate that the proposed VO algorithm is subject to more complex tests for evaluating its performance, and future works are needed to enhance its functionality.

	1 DoF test	2 DoF test	3 DoF test	Autonomous driving test
Total drift, meters	0.128	0.281	1.19	0.910
Total distance, meters	4.50	11.2	7.35	8.35
Accuracy rate	97.2%	97.4%	83.8%	98.9%

Table 4.2: Accuracy rate from teach test

Chapter 5: Conclusions and Future Work

This thesis presents an affordable solution for getting the global localization of an AUV. Different to other methods for localizing AUVs, the proposed approach doesn't require the developers to purchase expansive sensors, and it is portable for various platforms. In addition, by changing and optimizing the initial VO algorithm raised by Nister et al., we demonstrated that our VO algorithm is stable and accurate in cases the robot is moving in straight lines. We tested five feature detection algorithms on turbid underwater data set and found an algorithm with the best capability of extracting features from turbid underwater images. We chose to use that algorithm in the proposed VO algorithm, therefore, our system is considered to be robust in turbid underwater environments. The filter we added to the feature tracking part enables this VO algorithm to be more robust in dynamic underwater cases. At the same time, the other filter we added to the camera position estimation part reduces the accumulation of errors generated by the VO algorithm.

A few limitations have also been observed during tests. First, this algorithms suffers from the low accuracy rate while the robot is simultaneously rotating around the z axis and moving in straight lines. Second, the proposed algorithm has only be tested with robot motions on the xy-plane. It's ability of generating the trajectory of the robot while the robot moves in z direction or rolls/pitches has not been explored.

Finally, the primary focuses of future work for this research are listed below:

1. Establish a position estimation algorithm based on Kalman Filter for more accurate estimation of camera's position.

2. Conduct further research on implementing this VO algorithm for robot's moving cases including rotating around x and y axis and moving in the z direction.

3. Study the inertial based localization for AUV. Create another approach to estimate the localization of the robot based on the IMU data. Fuse the position estimation results from the visual approach and the inertial approach to generate a visual inertial odometry (VIO) algorithm.

4. Test the proposed VO algorithm in a real AUV to validate results from simulation and learn the sim-to-real transfer gap.

Bibliography

[1] Uwsim: Configuring and creating scenes. `https://www.irs.uji.es/uwsim/wiki/index.php?title=Configuring_and_creating_scenes`. Accessed: 2021-04-12.

[2] Mohammad OA Aqel, Mohammad H Marhaban, M Iqbal Saripan, and Napsiah Bt Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1–26, 2016.

[3] Kirk Baker. Singular value decomposition tutorial. *The Ohio State University*, 24, 2005.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[5] Patryk Cieslak, Pere Ridao, and Mariusz Giergiel. Autonomous underwater panel operation by girona500 uvms: A practical approach to autonomous underwater manipulation*. volume 2015, 05 2015.

[6] Amanda Duarte, Felipe Codevilla, Joel De O Gaya, and Silvia SC Botelho. A dataset to evaluate underwater image restoration methods. In *OCEANS 2016-Shanghai*, pages 1–6. IEEE, 2016.

[7] Maxime Ferrera, Julien Moras, Pauline Trouvé-Peloux, and Vincent Creuze. Real-time monocular visual odometry for turbid and dynamic underwater environments. *Sensors*, 19(3):687, 2019.

[8] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[9] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[11] Brian R. C. Kennedy, Kasey Cantwell, Mashkoor Malik, Christopher Kelley, Jeremy Potter, Kelley Elliott, Elizabeth Lobecker, Lindsay McKenna Gray, Derek Sowers, Michael P. White, Scott C. France, Steven Auscavitch, Christopher Mah, Virginia Moriwake, Sarah R. D. Bingo, Meagan Putts, and Randi D. Rotjan. The unknown and the unexplored: Insights into the pacific deep-sea following noaa capstone expeditions. *Frontiers in Marine Science*, 6:480, 2019.

[12] Pauline C Ng and Steven Henikoff. Sift: Predicting amino acid changes that affect protein function. *Nucleic acids research*, 31(13):3812–3814, 2003.

[13] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[14] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004.

[15] Alejandro Palmeiro, Manuel Martin, Ian Crowther, and Mark Rhodes. Underwater radio frequency communications. In *OCEANS 2011 IEEE-Spain*, pages 1–8. IEEE, 2011.

[16] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. Auv navigation and localization: A review. *IEEE Journal of oceanic engineering*, 39(1):131–149, 2013.

[17] Liam Paull, Mae Seto, Sajad Saeedi, and John J. Leonard. *Navigation for Underwater Vehicles*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.

[18] Mario Prats, Javier Perez, J Javier Fernandez, and Pedro J Sanz. An open source tool for simulation and supervision of underwater intervention missions. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 2577–2582. IEEE, 2012.

[19] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *European Conference on Computer Vision*, pages 500–513. Springer, 2008.

[20] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[21] Jianbo Shi et al. Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600. IEEE, 1994.

[22] Richard A Smith and Richard B Alexander. A statistical summary of data from the us geological survey's national water quality networks. *US Geological Survey Open-File Report*, 83(533):30, 1983.

[23] Jae Kyu Suhr. Kanade-lucas-tomasi (klt) feature tracker. *Computer Vision (EEE6503)*, pages 9–18, 2009.

[24] David Van Hamme, Werner Goeman, Peter Veelaert, and Wilfried Philips. Robust monocular visual odometry for road vehicles using uncertain perspective projection. *EURASIP Journal on Image and Video Processing*, 2015(1):1–21, 2015.

[25] Russell B Wynn, Veerle AI Huvenne, Timothy P Le Bas, Bramley J Murton, Douglas P Connelly, Brian J Bett, Henry A Ruhl, Kirsty J Morris, Jeffrey Peakall, Daniel R Parsons, et al. Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology*, 352:451–468, 2014.

[26] Ziv Yaniv. Median filtering. *School of Engineering and Computer Science The Hebrew University, Jerusalem, Israel*, 2009.

[27] Alan M Zhang and Lindsay Kleeman. Robust appearance based visual route following for navigation in large-scale outdoor environments. *The International Journal of Robotics Research*, 28(3):331–356, 2009.